VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017 Tomáš Král

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Král**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Railsformers s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Němec, Ph.D.**

Konzultant bakalářské práce: Ing. Richard Lapiš

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28.04.2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Railsformers s.r.o.
IČ: 24704440 DIČ: CZ24704440
www.railsformers.com
info@railsformers.com

V Ostravě 28.04.2017 .....

Rád bych na tomto místě poděkoval celé firmě Railsformers, s.r.o. za spolupráci. Zvláště Ing. Jiřímu Kubicovi, jednateli firmy, za umožnění vykonávat bakalářkou praxi. Dále pak Ing. Richardu Lapišovi, konzultantovi mé bakalářské práce, za ochotu a pomoc při řešení problémů a také celému kolektivu firmy, za profesionální přístup a příjemné pracovní prostředí.

Poděkování také patří vedoucímu mé bakalářské práce, Ing. Martinu Němcovi, Ph.D., za trpělivost a konzultaci při jejím vytváření.

V neposlední řadě bych rád poděkoval své rodině za jejich plnou podporu a umožnění mého studia na vysoké škole.

Abstrakt

Cílem této bakalářské práce je popsat mé působení u ostravské programátorské firmy Railsformers s.r.o., zabývající se primárně tvorbou webových aplikací pomocí Ruby a frameworku Ruby on Rails. V následujících kapitolách detailně popisuji a přibližuji průběh bakalářské praxe, jako je seznámení s novými technologiemi a praktikami, zadané úkoly a jejich řešení. Věnuji se také problémům, se kterými jsem se setkal a popisuji znalosti, které jsem se naučil, nebo které jsem získal při studiu na Vysoké škole báňské a následně je v praxi využil.

Klíčová slova: bakalářská práce, Ruby, Ruby on Rails, framework, web, HTML5, vývoj

Abstract

The aim of this bachelor thesis is to describe my work at the Ostrava programming company Railsformers s.r.o., mainly focused on creating web applications using Ruby and Ruby on Rails framework. In the following chapters I focus and describe in detail the process of bachelor practice, such as familiarizing with new technologies and practices, assigned tasks and their solutions. I also dedicate to problems that I have encountered and describe the knowledge I have learned or acquired during my studies at the VŠB and then used them in practice.

Key Words: bachelor thesis, Ruby, Ruby on Rails, framework, web, HTML5, development

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CRM	–	Customer Relationship Management
CSS	–	Cascading Style Sheets
CSV	–	Comma-separated values
DKIM	–	DomainKeys Identified Mail
DMARC	–	Domain-based Message Authentication, Reporting and Conformance
DOM	–	Document Object Model
ERB	–	Embedded RuBy
ETL	–	Extract, Transform, Load
HAML	–	HTML Abstraction Markup Language
HTML	–	HyperText Markup Language
HTTP(S)	–	HyperText Transfer Protocol (Secure)
IDE	–	Integrated Development Environment
IoT	–	Internet of Things
IPv4(6)	–	Internet Protocol version 4(6)
JSON	–	JavaScript Object Notation
MB	–	Megabajt
MVC	–	Model View Controller
MX	–	Mail Exchanger record
RAM	–	Random-access memory
REST	–	Representational State Transfer
RoR	–	Ruby on Rails
SAX	–	Simple API for XML
SPF	–	Sender Policy Framework
SQL	–	Structured Query Language
TSV	–	Tab-Separated Values
UI	–	User Interface
URL	–	Uniform Resource Locator
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce má za cíl popsat mé působení na odborné praxi v počítačové firmě Railsformers s.r.o. a zdokumentovat tak mou činnost ve firmě.

Pro praxi jsem se rozhodl, neboť jsem si chtěl vyzkoušet firemní prostředí, které se od toho akademického (spíše teoretického) liší, a zároveň si ověřit využitelnost znalostí nabytých při mém studiu na vysoké škole.

V následujících kapitolách přibližuji zajímavé projekty, na kterých jsem pracoval. V textu zmiňuji softwarové nástroje, metodiky vývoje a postupy, jež jsem využíval. Dále rozvádím problémy, se kterými jsem se musel vypořádat a popisuji jejich řešení. Zmiňuji, co nového jsem se naučil, a jak jsem v praxi využil znalosti z různých předmětů. Práci jsem rozdělil do několika částí:

- O firmě – krátké představení společnosti, u které jsem praxi vykonával
- Použité technologie a postupy – popsání metodik vývoje, návrhových vzorů a důležitých nástrojů
- Projekty – popis jednotlivých projektů na kterých jsem pracoval, použitých knihoven, problémů a jejich řešení
- Závěr

V konečném důsledku doufám, že práce přinese nejen popis mé bakalářské praxe, ale pomůže i budoucím začínajícím programátorům se zájmem o programování v RoR, k nalezení zajímavých nástrojů a popisu řešení problémů, na které by mohli, stejně jako já, narazit.

2 O firmě

Railsformers s.r.o. jsou ostravskou počítačovou firmou, která byla založena roku 2012 Jiřím Kubicou. Rozsahem a fungováním se řadí spíše k malým podnikům [1].

Primárně se zabývají vývojem webových aplikací a informačních systémů. Specializují se na projekty větších rozsahů jako jsou komunitní portály, enterprise řešení pro firmy, vývoj komplexních systémů a smartcity projekty. Dále se zabývají tvorbou mobilních aplikací pro Android a iOS, prací s velkými daty a komunikací a dorozumíváním v IoT.

Pracují na zakázkách jak pro domácí, tak pro zahraniční klienty. Kladou velký důraz na SEO optimalizaci, rychlé se přizpůsobení novým technologiím, a jejich následné používání v praxi.

Mezi hlavní softwarové projekty firmy patří:

1. Hedurio – nejvýznamnější projekt firmy na bázi cloudového řešení, jedná se o online informační systém pro řízení bezpečnostních agentur
2. ELUC – elektronická učebnice pro výuku na středních školách v Olomouckém kraji
3. JOKRYS – vzdělávací systém a e-knihovna pro školní vzdělávací plány
4. sÚčto – aplikace pro správu faktur, jejich šablon a online účetnictví
5. sRecepty – portál zabývající se zdravou výživou a vším kolem jídla, obsah tvoří především uživatelé
6. Reservatic – online rezervační systém sloužící k rezervaci služeb

Firma samozřejmě spravuje a vyvíjí mnoho dalších projektů, které má uvedeny na svých webových stránkách [2].

3 Použité technologie a postupy

Důležitou součástí návrhu a tvorby softwaru je výběr správného programovacího jazyka a dalších nástrojů. Pro tvorbu backend webových aplikací se nejčastěji používají buď staticky typované jazyky, jako je Java nebo C#, ale velké oblibě se těší i jazyky, které jsou typované dynamicky a umožňují velmi rychlou a snadnou tvorbu softwaru. Příkladem takových jazyků může být PHP, Python, Perl, Javascript nebo třeba Ruby.

3.1 Ruby

Ruby je dynamický, interpretovaný, objektově orientovaný, skriptovací programovací jazyk, který je díky své jednoduché syntaxi poměrné snadný k naučení.

Tvůrcem jazyka je jediný člověk Jukihiro Macumoto. Nejpodobnějšími jazyky k Ruby jsou Python nebo Perl, ale žádný z nich není tolik objektový, neboť v Ruby je všechno objekt [3].

3.2 Ruby on Rails

Autorem je dánský programátor David Heinemeier Hansson, který ho publikoval v roce 2004. Jedná se o open-source framework pro rychlý a pohodlný vývoj internetových aplikací v jazyce Ruby, který využívá architekturu Model-View-Controller. Dnes ho vylepšují tisíce přispěvatelů, ať už pomocí gemů nebo přispíváním do frameworku. Mezi nejznámější projekty využívající Rails patří Github, Twitter, Twitch, Shopify, AirBnb, SoundCloud, Kickstarter a mnoho dalších [4].

3.3 Gemy

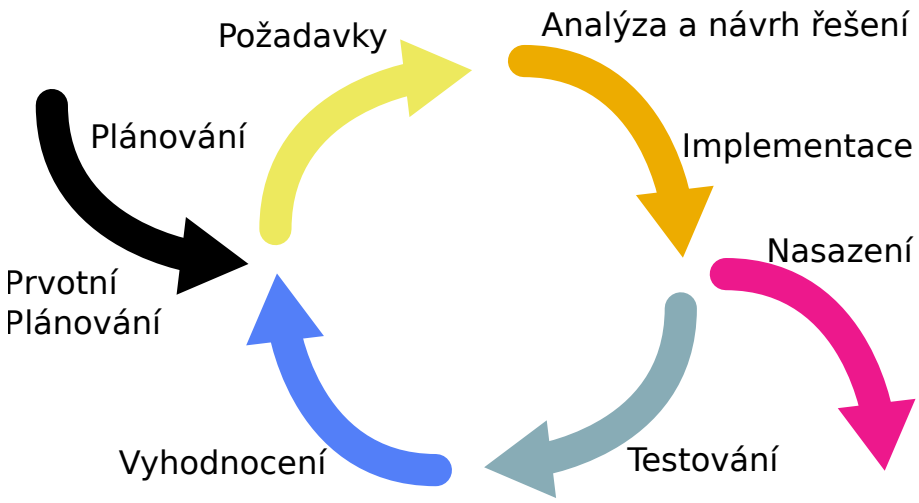
Gem je standardní formát pro distribuci programů a knihoven. Je to vlastně systém správy balíčků pro Ruby. Gemy pochází buď od vývojářů Ruby, RoR nebo od dalších programátorů. Pomocí těchto balíčků si programátor ulehčuje práci. Vývojář může přidat gem buď pro celý systém nebo pouze pro danou aplikaci v Ruby on Rails, v tom případě je ale nutné daný gem uvést v *Gemfile* [5].

3.4 Agilní programování a inkrementální přístup

Ve firmě se nejvíce využívají dvě metodiky vývoje. Primárně se uplatňují principy agilního programování v kombinaci s inkrementálním přístupem. S těmito metodikami tvorby softwaru, jsem byl již seznámen v rámci předmětu Úvod do softwarového inženýrství (SWI) a využíval jsem je i v průběhu mé odborné praxe [7].

Zadání projektu jsem si rozdělil do menších částí (iterací), které jsem vždy implementoval a pokračoval prací na další části. Důraz byl kladen na rychlé dodávání nových funkcí v častějších intervalech a po menších částech a také na jednoduchost (kódu a vývoje). Zároveň byly neustále aktualizovány a přidávány další požadavky na projekty. Pokud byla potřeba, komunikoval jsem osobně s různými členy týmu.

V praxi jsem si tak vyzkoušel, jak jsou tyto techniky účinné a dovedou rychle reagovat na změny v požadavcích zákazníka. Agilní techniky v kombinaci s RoR, přináší výborný způsob tvorby kvalitního softwaru v krátkém čase.



Obrázek 1: Iterativní proces [8]

3.5 Architektury a návrhové vzory

Filosofie Rails aplikace obsahuje několik vůdčích principů. Prvním je *DRY* (Don't Repeat Yourself) princip, který říká, že psát stejný kód stále dokola je špatné.

Dalším principem je *convention over configuration* (konvence má přednost před konfigurací) a znamená, že Rails předpokládá co asi chce programátor udělat a jak to chce udělat, místo aby ho nutil specifikovat každou drobnost v konfiguračních souborech. Proto je kladen důraz především na strukturu projektu a názvosloví.

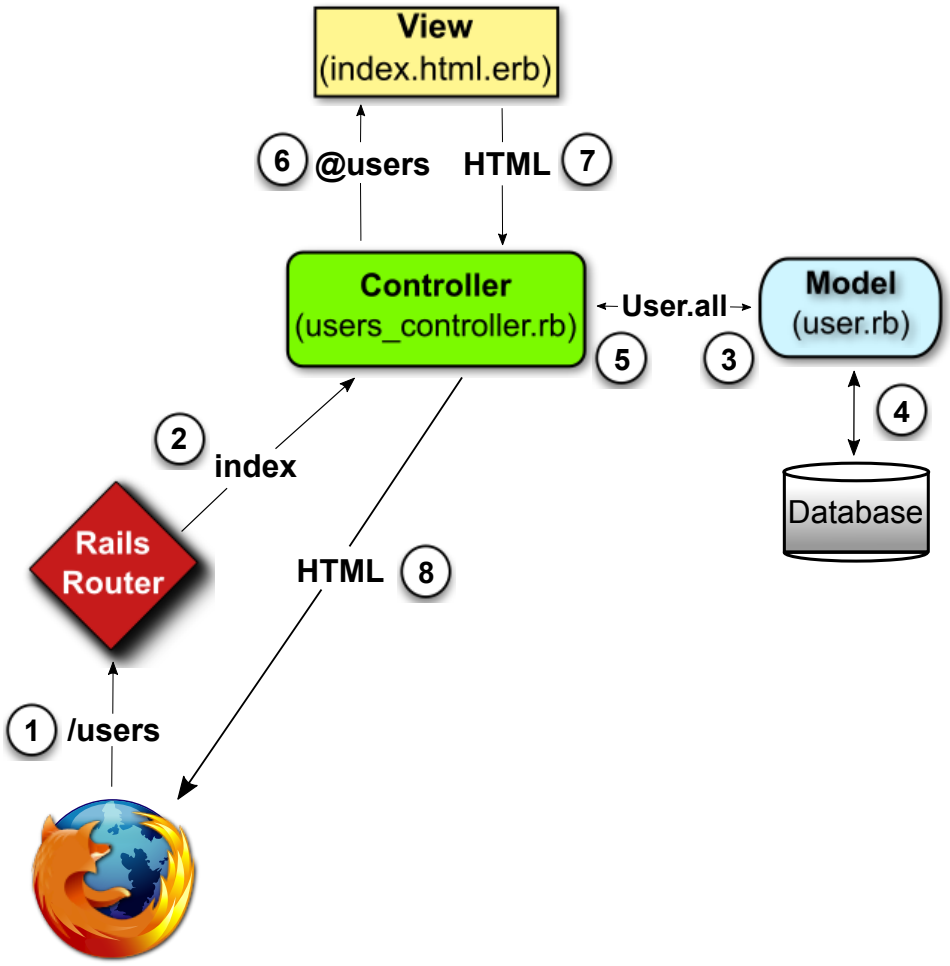
RoR je postaven jako REST framework, což je architektonický vzor, který uspořádává webovou aplikaci jako soubor zdrojů. Rozhraní REST je použitelné pro jednoduchý a snadný přístup ke zdrojům a je to jeden z nejsnazších a nejrychlejších způsobů, jak modelovat entity a jejich vztahy.

3.5.1 Model-View-Controller (MVC)

RoR je založeno na architektuře Model-View-Controller. Princip zpracování požadavku klienta je zřejmý z obrázku a je poměrně podobný zpracování požadavku ve frameworku Django, se kterým jsem se setkal v rámci předmětu Skriptovacích a programovacích jazyků.

- Model – obsahuje většinou aplikační logiku a reprezentuje informace, s nimiž aplikace pracuje
- View (pohled) – převádí data reprezentovaná modelem do podoby pro prezentaci uživateli

- Controller – reaguje na události, zajišťuje změny a propojuje modely a pohledy



Obrázek 2: Implementace MVC v Ruby on Rails [12]

3.5.2 Active Record

Představuje model v Rails aplikaci a je zodpovědný za reprezentaci byznys dat a logiky. U Active Record si objekty s sebou nesou jak data, tak i chování operující nad daty. V RoR reprezentuje vztahy mezi modely, dědičnou hierarchii, ověřuje data před jejich uložením do databáze a díky objektově relačnímu mapování, zapouzdřuje databázové operace jako práci s objekty.

3.6 ERB a HAML

ERB je výchozí formát pro views v RoR. Jedná se o HTML dokument s vloženým Ruby kódem uvnitř, který se vykoná, respektive vyhodnotí.

HAML je značkovací jazyk, který slouží jako alternativa k ERB. Jeho největší výhodou je snadnější zápis, větší přehlednost a daleko rychlejší psaní dokumentů. Nevýhodou může být

ze začátku zvyk na nový způsob zápisu a také zvyk na odsazení jednotlivých položek dokumentu. Nejlépe je možné si rozdíly ukázat na příkladu.

```
<section class="container">
  <h1><%= post.title %></h1>
  <h2><%= post.subtitle %></h2>
  <div class="content">
    <%= post.content %>
  </div>
</section>
```

Výpis 1: Ukázka kódu v ERB

```
%section.container
  %h1= post.title
  %h2= post.subtitle
  .content
    = post.content
```

Výpis 2: Ukázka stejného kódu v HAML syntaxi

HAML sází na často používané elementy v HTML dokumentu a zjednodušuje jejich zápis. Díky odsazování se nemusí jednotlivé tagy ukončovat. Navíc je syntaxe velmi snadná k naučení a opravdu zvyšuje přehlednost, rychlost a produktivitu při psaní šablon HTML stránek.

Pro použití HAMLu v RoR projektu stačí nainstalovat gem **haml** a následně přejmenovat views s příponou *.html.erb* na *.html.haml* [9].

3.7 Další použité nástroje

Vývoj v Ruby on Rails je nejjednodušší provádět na nějakém Unix-like systému. Je to z toho důvodu, že mnoho gemů využívá příkazy typické pro Unixovou příkazovou řádku a vývoj na Windows může být zdlouhavý, kvůli neustálému řešení problémů s kompatibilitou. Jelikož jako primární systém používám Ubuntu, nepředstavovalo pro mě tohle „omezení“ problém.

Trochu těžší bylo vybrat nástroj pro vývoj v RoR. Většina komunity využívá jednoduché textové editory jako je SublimeText, Atom nebo TexMate. IDE nástrojů moc není, ale známým a často používaným je Rubymine od JetBrains. Licence je poměrně drahá, ale studentům JetBrains poskytuje své produkty zdarma. Nakonec mi nejvíce vyhovovaly editory **SublimeText** [10] pro svoji rychlost a jednoduchost a IDE **Rubymine** [11], které zase poskytuje mnoho nadstandardních možností práce s kódem.

3.8 Harmonogram praxe

Při nástupu na praxi bylo mým prvním úkolem, nainstalovat si správné verze Ruby, RoR a Bootstrapu, zprovoznit firemní email a seznámit se s *Projektově*, což je systém, který slouží pro plánování a správu úkolů ve firmě, a který je rovněž jedním z produktů firmy [2].

Díky knize Michaela Hartla [12], jsem nebyl ve vývoji aplikací v RoR úplným začátečníkem a znal jsem základní syntaxi a organizaci projektu, a proto jsem byl zařazen mezi Rails vývojáře. Ze začátku jsem si také musel nastudovat mnoho gemů, které ve firmě používaly, a které jsem neznal.

V průběhu praxe jsem pracoval celkově na třech různých projektech. Projekty jsem seřadil do kapitol chronologicky, podle toho, kdy jsem na daném projektu začal pracovat. Zhruba více jak polovinu praxe mi zabrala práce na prvním projektu, zbytek potom vypracování ostatních dvou projektů, které mi zabralo u každého víceméně stejný čas.

Tabulka 1: Přehled přibližné časové náročnosti jednotlivých bodů praxe

Projekt	**Typ práce**	**Čas (ve dnech)**
RfCheck aplikace	Sidekiq	7
	API	6
	ostatní funkce	7
	řešení chyb	5
CRM systém	studium	4
	implementace funkcí	7
TopZájezdy	OpenRefine	6
	ETL	4
	parsování XML	4

4 RfCheck

RfCheck byl první aplikací, který jsem dostal za úkol vytvořit. Ze začátku mi byla zadána pouze základní funkčnost, později však byly postupně přidávány další funkční a ergonomické požadavky. Z tohoto důvodu jsem se rozhodl, že v první části kapitoly popíšu základní implementaci a v té druhé se pak budu věnovat vývoji dalších rozšíření a funkcí, postupně přidávaných do aplikace.

4.1 Zadání práce

Prvotní zadání bylo vytvořit aplikaci, která bude sloužit pro interní potřebu firmy ke kontrole různých věcí. Aplikace měla primárně zvládat tyto úkony:

- U seznamu URL adres kontrolovat jejich status (odpovídá, neodpovídá).
- U IP adres kontrolovat, zda odpovídají na ping a zda daná IP není vedena na nějakém veřejném blacklistu.
- U HTTPS adres kontrolovat platnost a datum vypršení certifikátu.
- U seznamu domén kontrolovat záznamy MX, SPF, DKIM, DMARC.

Při nějakém výpadku či změně statusu, ať už URL, IP adresy nebo při skončení platnosti certifikátu, měla aplikace upozornit uživatele zasláním emailu. Dalším požadavkem bylo vytvoření algoritmu, aby aplikace nezahlcovala servery neustálými požadavky moc často, a aby se kontrola jednotlivých položek měnila podle předchozí spolehlivosti, respektive nespolehlivosti dané adresy (např. pokud byla daná adresa v minulosti spolehlivá, čas její kontroly se prodlouží). Kontrola jednotlivých položek měla probíhat pomocí delayed job (viz kapitola 4.4). Do aplikace měli mít přístup pouze přihlášení uživatelé. Jelikož firma Railsformers spolupracuje s dalšími společnostmi, mělo být v aplikaci možné přidávat další firmy. Každá firma potom měla mít svůj seznam uživatelů a adres, uživatelé měli mít možnost, se mezi jednotlivými firmami přepínat. Celá aplikace měla běžet na nové verzi RoR 5.0.

Později se k prvotním požadavkům na aplikaci přidali ještě další:

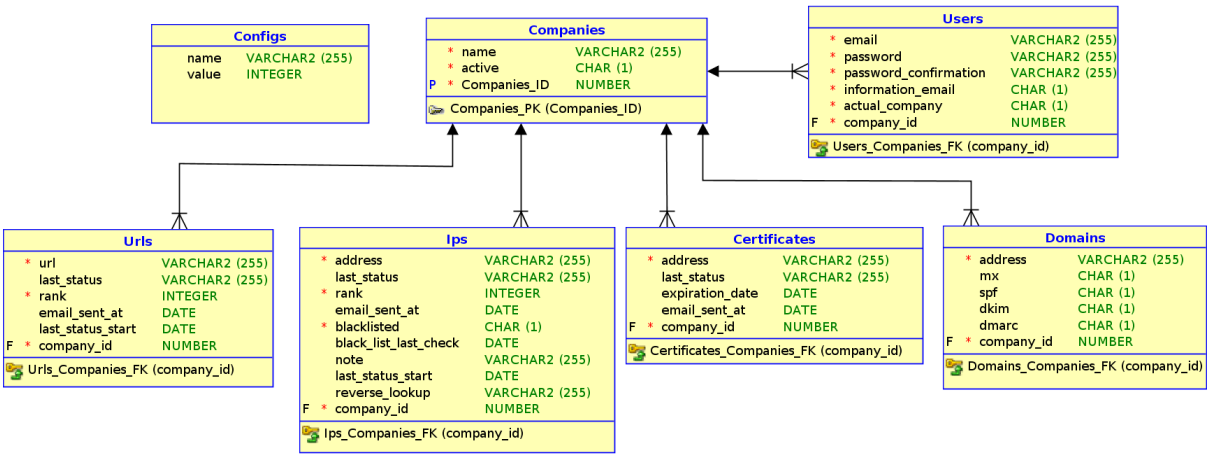
- Vytvořit API, které by umožňovalo přístup k datům dalším aplikacím (především mobilním).
- U IP adres kontrolovat DNS záznam.
- U domén kontrolovat, kdy platnost domény vyprší a dát informaci o SEO optimalizaci, jako jsou hodnoty různých ranků, počty zpětných odkazů a indexů u vyhledávačů apod.
- Dodělat stránku s výpisem z poštovního serveru (log odeslaných emailů ve firmě).
- Přidat podporu pro zasílání notifikací na telefony.
- Seskupovat více emailů a notifikací do jedné zprávy.

4.2 Vlastní práce na projektu

Jelikož je tohle nejrozsáhlejší kapitola a popsat vše by bylo nad rámec rozsahu této práce, zaměřil jsem se na nejzajímavější technologie a problémy, které jsem v průběhu vývoje musel řešit.

4.2.1 Analýza zadání

Nejprve bylo nutno analyzovat požadavky zadání projektu. Navrhl jsem, jak bude databáze vypadat a jaké všechny informace bude o daných položkách ukládat.



Obrázek 3: Relační model databáze pro RfCheck

Z uvedeného relačního modelu je zřejmé, k čemu slouží jaká tabulka. Urls ukládá informace o jednotlivých kontrolovaných URL adresách, Ips zase o IP adresách. Certifikates slouží pro ukládání HTTPS adres, u kterých má být kontrolována platnost certifikátu. Domains tabulka udržuje informace o doménách, kde se kontrolují MX, SPF, DKIM a DMARC záznamy. Tabulka Configs, slouží pro ukládání konstant a nastavení použitých napříč aplikací. Jeden záznam v tabulce odpovídá jednomu nastavení v aplikaci. Jednotlivé hodnoty lze měnit v menu v položce nastavení. Jelikož se jedná o interní aplikaci, je nastavení implementováno globálně. Companies reprezentuje jednotlivé firmy v systému a Users odpovídá uživatelům, kteří se mohou do aplikace přihlásit.

4.2.2 Vytvoření modelů

Jelikož požadavek na rozdělení uživatelů a položek do různých společností, byl přidán až později, začal jsem tvorbou modelů s názvy Url, Ip, Certificate, Domain a jejich příslušných validací.

```
require 'uri'
```

```ruby
class Url < ApplicationRecord
  belongs_to :company
  before_save :set_default_values
  before_validation :edit_url
  validates :url, presence: true, length: { maximum: 400 }, :format => URI::
     regexp, :url => true
  validates_uniqueness_of :url, scope: :company_id
  validate :valid_url

  def respond?
    return self.last_status.to_i.between?(200, 299)
  end

  private
    def set_default_values
      self.rank ||= Config.find_by_name("URL_MIN_CONTACT_TIME").value
    end

    def valid_url
      errors.add(:base, 'Invalid HTTPS address') unless !!URI.parse(self.url)
    end
    ...
end
```

Výpis 3: Výpis modelu Url

Vztah mezi Urls a Companies je zapsán pomocí *belongs_to* v modelu Url a *has_many* v modelu Company - tímto se v RoR vyjádří vztah mezi oběma modely. *before_save* je callback, který se provede před uložením dat do databáze. Zde pomocí tohoto callbacku nastavím výchozí hodnotu ranku, která se vezme z tabulky Config. Před tím, než se objekt zapíše do databáze se provádí validace. U URL validuji délku, která nesmí přesáhnout 400 znaků, *presence: true* ověřuje zda je daný atribut vyplněný a pomocí *format* zjišťuji, zda se jedná o validní URL adresu. Dále testuji, zda je daná URL unikátní v rámci firmy, aby nedocházelo ke vkládání stejných záznamů.

Modely Ip, Certificate a Domain mají velmi podobné validace. U IP adresy se ověřuje validnost dané adresy pomocí třídy *IpAddress.valid?*, která dovede testovat validitu IPv4 i IPv6 adres.

Přihlašování uživatelů jsem udělal pomocí gemu **Devise** [13]. Ten je v RoR jedním z nejoblíbenějších pro řešení registrace, autentizace a ověřování uživatelů, práce s ním je jednoduchá a existuje pro něj mnoho dokumentace a návodů.

4.3 Uživatelské rozhraní a funkcionalita

Pro tvorbu uživatelského rozhraní jsem použil Bootstrap, což byl také jeden z požadavků. Bootstrap je jeden z nejpopulárnějších frameworků pro tvorbu responzivních webových projektů pomocí HTML, CSS a JavaScriptu. V rails aplikaci stačí přidat **bootstrap** gem a v *application.css.scss* ho naimportovat. Následně lze ve views již využívat všech předpřipravených tříd a funkcí Bootstrapu. Díky široké škále tříd, které poskytuje, jsem za celou dobu tvorby aplikace nemusel psát skoro žádné CSS styly. Výjimkou byla stylizace emailů, v jejichž view šablonách se Bootstrap třídy použít nedají.

Při zvyšujícím se počtu položek v seznamech se také vyskytl požadavek na hledání mezi nimi. Tento problém jsem vyřešil pomocí gemu **Ransack**. Ten umožňuje řadit položky podle různých kritérií a parametrů, a zároveň mezi nimi vyhledávat. Pro jeho funkčnost stačí v příslušném controlleru upravit dotaz do databáze a ve view potom použít pomocné metody Ransacku, které tento gem přidává.

Pro stránkování jsem použil gem **Kaminari**. Pomocí něj lze stránkovat položky a z databáze se načítají data vždy pouze pro aktuální stránku. Použití je velmi jednoduché, neboť tento gem přidává parametr *page*, který stačí přidat do databázového dotazu do příslušné akce controlleru.

```
def set_variables
    @q = Url.where(company_id: current_company.id).order("id DESC").ransack(
       params[:q])
    @urls = @q.result.page params[:page]
end
```

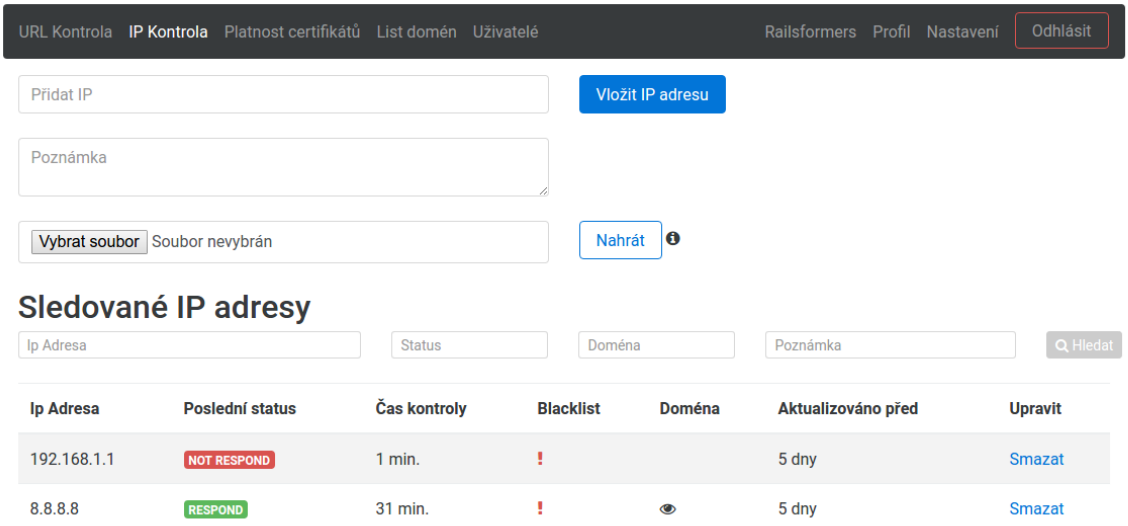
Výpis 4: Ukázka použití Ransack a Kaminari gemů

Dalším zajímavým gemem, který mi pomohl při tvorbě uživatelského rozhraní, je **font-awesome-rails**. Jedná se o vektorové ikony, které jsou ve skutečnosti písmem a přebírají všechny jeho vlastnosti. Díky tomu lze upravovat jejich styly pomocí CSS jako u normálních písem a ze serveru se nemusejí stahovat rastrové obrázky. Ty by navíc nemusely být tak ostré na různých rozlišeních a nepracovalo by se s nim tak pohodlně. Do views potom přidává helper *fa_icon*.

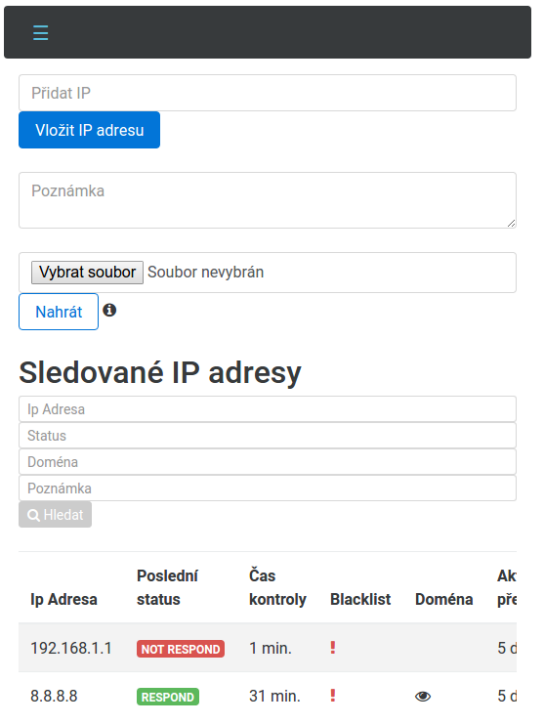
4.4 Delayed job, Sidekiq a zpracovávání na pozadí

Delayed job slouží k provádění úkolů na pozadí a umožňuje jejich řazení do fronty. V RoR se nejčastěji používá např. pro odesílání emailů, práci s databází, jednoduše pro veškeré činnosti, které nepotřebujeme hotové ihned, ale mohou se vykonávat asynchronně na pozadí.

Sidekiq je obdobou delayed job, ale je daleko rychlejší a efektivnější při práci s pamětí a při vykonávání jobů (viz tabulka 2). Z tohoto důvodu jsem jej po konzultaci s mým vedoucím ve firmě, zvolil jako hlavní nástroj pro vykonávání jobů na pozadí. Nejdříve jsem si nastudoval dokumentaci Sidekiq, abych lépe pochopil, jak v něm úkoly na pozadí fungují a jak se řadí do front. Pro uchovávání operačních dat, jobů a ostatních věcí využívá **Redis** [14].



Obrázek 4: Ukázka vzhledu aplikace RfCheck



Obrázek 5: Ukázka vzhledu aplikace RfCheck na mobilním zařízení

Joby/Workery pro Sidekiq lze v RoR vytvořit dvěma způsoby:

1. Vytvoříme složku *app/workers* kde vytvoříme tzv. workery. Sidekiq si potom sám načte tuhle složku a wokery uvnitř lze plánovat a vykonávat.

2. Druhou možností je vytvořit pro rails standardní delayed job v *app/jobs*.

V *config/application.rb* se však potom musí nastavit adaptér na Sidekiq (viz výpis 5). Díky tomu lze psát úkoly na pozadí jako delayed job, ale budou se vykonávat pomocí Sidekiqu.

```
config.active_job.queue_adapter = :sidekiq
```

Výpis 5: Nastavení Sidekiq jako adaptéru pro vykonávání delayed jobs

```
...
class IpsWorker < GeneralWorker
  sidekiq_options retry: false
  sidekiq_options unique: :until_and_while_executing

  def perform(ip_id)
    Raven.capture do
      @ip = Ip.find_by_id(ip_id)
      if !@ip.nil? && @ip.company.active
        status = @ip.last_status || "RESPOND"
        ip_check
        IpsWorker.perform_in(@ip.rank.minutes, @ip.id)
        send_status_changed_email(@ip, status) if !status.nil? && status != @ip
            .last_status && @ip.last_status == "RESPOND"
      end
    end
  end
   ...
end
```

Výpis 6: Ukázka workeru IpsWorker pro kontrolu IP adres

Ze začátku jsem zvolil první možnost, ale později kvůli chybě (viz kapitola 4.8), jsem workery přepsal do delayed job. Workery jsem vytvořil ve složce *app/workers/*. Každý worker musí obsahovat veřejnou metodu **perform**. Tato metoda je automaticky volána, jakmile je daný pracovník spuštěn. Jak lze vidět v kódu, worker se provede, a rekursivně naplánuje své znovu-spuštění pomocí *IpsWorker.perform_in(za-jak-dlouho, id-ip-adresy)*.

Každá kontrolovaná položka, ať už IP nebo URL adresa, má svůj tzv. rank. Rank značí, jak byla daná položka v minulosti spolehlivá. Při spolehlivosti se rank navyšuje, při výpadku se resetuje na minimální hodnotu. Minimální a maximální hodnota ranku je určena pomocí záznamů v tabulce Config. Další práci, kterou tento worker provádí je, že v případě výpadku zašle email všem uživatelům, kteří jsou k odběru emailů přihlášeni.

Většina aplikací ve firmě využívá **Sentry**, což je moderní systém pro zachytávání chyb. Díky *Raven.capture do* jsou veškeré chyby, výjimky a další nedostatky zaznamenány, a jako vývojář jsem o nich informován [15].

Tabulka 2: Srovnání Sidekiq s podobnými nástroji

Verze	**Latence**	**Odpad vytvořený pro 10,000 jobů**	**Čas potřebný k vykonání 10,000 jobů**	**Propustnost**
Sidekiq 4.0.0	10 ms	151 MB	22 sec	4500 jobs/sec
Sidekiq 3.5.1	22 ms	1257 MB	125 sec	800 jobs/sec
Resque 1.25.2	-	-	420 sec	240 jobs/sec
DelayedJob 4.1.1	-	-	465 sec	215 jobs/sec

Problém, který může nastat je, že při vykonávání workeru dojde k chybě a takový worker by již nestihl naplánovat své opakované spuštění. Vedlo by to k tomu, že by se objevovaly položky, které by již nebyly kontrolovány a aktualizovány, protože jejich workery na pozadí již neběží. Z tohoto důvodu jsou veškeré workery pomocí Cronu v pravidelném intervalu znovu-spouštěny.

Cron je softwarový démon, který automatizovaně v operačních systémech spouští v určitý čas nějaký příkaz. V mém případě jsem musel vytvořit rake task (skript, který lze volat z RoR aplikace) ve složce *lib/tasks/*. Tento rake task spouští workery pro všechny položky v databázi. Díky gemu **sidekiq-unique-jobs**, se naplánují a spustí workery pouze pro ty položky, pro které ještě naplánované nejsou. To zamezuje tomu, aby dva a více pracovníků kontrolovali tu samou položku, což by bylo velmi neefektivní a mohlo by to vést k deadlockům. Zde tuto funkcionalitu zajišťuje klauzule *:until_and_while_executing*, kterou přidává právě zmiňovaný gem.

4.5 Tvorba API

V druhé fázi tvorby aplikace jsem dostal za úkol vytvořit k aplikaci API. API mělo umět poskytovat data z databáze ve formátu JSON, ale pouze autorizovaným uživatelům. Vytvořit API v RoR jde více způsoby. Od verze Rails 5.0 ho lze tvořit přímo bez potřeby nějakých rozšíření. Další možností je použití gemu **Grape**. Jedná se o REST-API framework, který funguje jak pro čisté Ruby, tak pro frameworky jako Rails. Jelikož se mi líbila syntaxe Grape gemu a dokumentace byla rozsáhlá, zvolil jsem tuto možnost [16].

Ze začátku mi psaní API pomocí Grape nepřipadalo složité. Vytvořil jsem složku *api/* ve složce *app/controllers/* a zde potom strukturu podle dokumentace. Následně jsem přistoupil k psaní samotného API.

```
module API
  module V1
    class Certificates < Grape::API
      include API::V1::Defaults

      resource :certificates do
        before do
```

```
      authenticate_user!
    end

    desc 'Return all certificates for actual company'
    get do
      status 200
      present certificates: paginate(Certificate.where(company_id:
          current_user.actual_company))
    end
...
```

Výpis 7: Ukázka API pro certifikáty v Grape

Díky *desc* popisům u každé metody je zřejmé k čemu slouží. Metoda *get* na adresu *domain/api/v1/certificates* nejprve pomocí callbacku *before_do* ověří, zda je daný uživatel přihlášen a následně mu vrátí JSON odpověď s certifikáty a status 200. *Paginate* zde zajišťuje gem **grape-kaminary**, který se stará o stránkování v API pro Grape a používá se stejně stejně jako Kaminari ve views (viz kapitola 4.3).

Hlavním problémem, který mi zabral hodně času byla autorizace uživatele. K zajištění této funkcionality slouží gem **grape-token-auth**, který má ovšem velmi omezenou dokumentaci. Zde jsem strávil nejvíce času zjišťováním, jak se vůbec může uživatel autorizovat a jaké parametry musí posílat v JSON požadavku, aby ho systém nezamítl. Většinou nezbylo nic jiného, než se přímo podívat do kódu daného gemu, pochopit jak funguje a z toho následně určit jaké parametry jsou potřeba.

Do kódu gemu se dá dostat například pomocí příkazu *bundle show [název gemu]*. Při této práci mi byl velmi nápomocný gem **byebug**, který slouží k ladění aplikací v ruby. Kdekoliv v kódu stačí napsat byebug a jakmile se začne daný kód vykonávat, aplikace se zastaví a lze ji krokovat, zjišťovat obsah proměnných a provádět další činnosti typické pro ladění. Díky tomu jsem zjistil, že k autorizaci je potřeba posílat s požadavkem tři údaje v hlavičce a to:

- access-token – jednoznačný identifikátor daného zařízení
- uid – jedinečné Id uživatele
- client – v mém případě email uživatele, ale je možné nastavit

Tyto údaje dostane uživatel v JSON odpovědi při autentizaci. Jestliže je jeden z těchto údajů neplatný, server vrací stavový kód 401 (Unathorized), jinak vrací 200 (OK) a požadovaná data v JSON formátu. Ve výchozím stavu je navíc nastavené, že při každém požadavku na server, se hlavičky mění. Tohle chování jsem vypnul za cenu nižší bezpečnosti, ale jednoduššího používání API.

Autentizaci uživatelů již navíc řešil gem Devise. Ten používá některé stejné pomocné metody jako grape-token-auth. Proto, aby se tyto gemy nepřekrývaly a byla zajištěna bezproblémová funkčnost, bylo nutno doinstalovat a nastavit **devise_token_auth** gem a **grape_devise_token_auth** gem.

4.6 RSpec, testování a dokumentace

Jakmile jsem měl vytvořené API, bylo nutno pro něj napsat dokumentaci a otestovat jeho funkčnost. V Rails aplikaci lze testovat buď pomocí vnitřního testovacího prostředí anebo pomocí **RSpec** gemu, který poskytuje daleko větší škálu funkcí a u RoR programátorů je oblíbenější [17]. Z jeho používání plynula navíc další výhoda a to, že z těchto testů bylo možné následně automaticky vygenerovat dokumentaci pomocí gemu **Apitome**. Tím jsem zajistil jak otestování funkčnosti API, tak i vygenerování příslušné dokumentace.

```
...
delete 'api/auth/sign_out' do
  example 'Sign out user' do
    header 'access-token', token
    header 'uid', uid
    header 'client', client_id
    do_request( {} )
     expect(json_response['success']).to eq(true)
      expect(status).to eq 200
    end
  end
end
...
```

Výpis 8: RSpec ukázka testu odhlášení uživatele

Testy musí být umístěny ve složce *spec/acceptance/*. Apitome následně spustí testy a pokud jsou psány tímto způsobem (viz výpis 8), vygeneruje z nich dokumentaci. Ta je potom přístupná na adrese *domain/api/docs/*. Problém, který jsem musel řešit byl, že dokumentace byla přístupná i nepřihlášeným uživatelům. Řešení tohoto problému bylo jednoduché. Stačilo vytvořit controller potomka, který dědí z Apitome::DocsController a přidat callback pro autentizaci uživatele. V *routes.rb* pak stačilo přidat tuhle cestu.

4.7 Implementace ostatních funkcí

Při práci na dalších funkcích jsem postupoval, stejně jako předtím, podle agilních metodik. Po implementaci jsem kladl důraz na otestování nové funkcionality a vždy až potom jsem se

přesunul na další úkol. I když jsem implementoval daleko více funkcí, zmiňuji zde ty, které byly nejzajímavější nebo byly časově náročnější k naprogramování.

4.7.1 Vypršení platnosti a hodnocení domény

Konec vypršení platnosti registrace domény se nejlépe zjišťuje pomocí příkazu *whois*. V Ruby na tohle existuje gem **whois-parser**, který dokáže vrácená data z whois příkazu vytáhnout.

Trošku větším problémem se ukázalo zjišťování ranků a dalších informací o doméně z různých vyhledávačů. Každý vyhledávač určuje rank stránky podle odlišných kritérií (SEO optimalizace, počet odkazů atd.). Pro začátek mi dobře posloužil gem **page_rankr**. Ten dovede zjistit počet zpětných odkazů na danou doménu a počet indexů při zadání do různých vyhledávačů. Dříve dovedl najít také pagerank od Google, ale ten byl ze strany společnosti Google zrušen. Díky tomuto gemu jsem tak dovedl zjistit počet zpětných odkazů a indexů z vyhledávačů Google a Bing.

Dále jsem využil **linkscape** gem, který dovede zjišťovat rank a počty zpětných odkazů z MozRanku, který měří rank stránek na základě kvality odkazů, které na ně vedou.

Nepatrně složitější bylo zjištění ranku (tzv. sranku) ze Seznamu. Seznam je v České republice důležitý vyhledávač, a tudíž vědět, jak si stránky spadající pod firmu stojí v jeho ranku, se ukázalo jako důležitý požadavek. Seznam neposkytuje žádný návod, jak se k jeho sranku dostat. Na internetu jsem však našel skripty napsané v Pythonu a PHP, které dokázaly zjisti srank, pomocí XML-RPC volání na server. Musel jsem tedy dané skripty vyzkoušet a následně přepsat do Ruby. Jelikož jsou Ruby a Python velmi podobné jazyky, nebylo to moc složité, ovšem obě knihovny se v obou jazycích používají trochu jinak. Daný kód potom vrací srank hodnotu od 0 do 10 (viz výpis 9).

```ruby
def count_seznam_srank
  server = XMLRPC::Client.new2('http://srank.seznam.cz')
  begin
    result = server.call('getRank', '0', @domain.address, 0)
  end
  result['rank'] == 0? nil : ((result['rank']/2.55).round.to_i/10)
end
```

Výpis 9: Ukázka zjištění Sranku pomocí XML-RPC

Seznam bohužel neposkytuje žádné API ani cestu jak zjistit počet zpětných odkazů nebo indexů, takže jsme se museli, v tomto případě, spokojit jenom s jeho konečným hodnocením dané domény.

4.7.2 Data z poštovního serveru

Následujícím požadavkem bylo zobrazování dat z firemního poštovního serveru v mé aplikaci. Jelikož již pro tento účel byla napsána webová stránka v konkurenčním frameworku Sinatra [19], stačilo se připojit ke vzdálené databázi, získat data a přepsat view do mé aplikace v RoR. Připojení ke vzdálené databázi se řeší buď pomocí přidání konfigurace do *database.yml* nebo pomocí *connection string*. Pro každou tabulku vzdálené databáze, pokud s ní chceme pracovat pomocí ActiveRecord, je nutno vytvořit model.

```
class PostfixLog < ApplicationRecord
  establish_connection(:postlog)
  enum delivery_success: { yes: 1, no: 2, soft_bounce: 3 }
  self.table_name = 'postfix_logs'
end
```

Výpis 10: Ukázka modelu pro vzdálenou databázi

Díky klauzuli *establish_connection* Active Record ví, že pro daný model se má připojovat na vzdálenou databázi a díky *self.table_name* se dozví, jakou tabulku z dané databáze reprezentuje.

4.7.3 Podpora notifikací

Velmi zajímavým úkolem bylo vytvořit podporu pro zasílání notifikací na chytrá zařízení. S konzultantem praxe jsme se domluvili, že navíc nebudu zprávy posílat po jedné jako doposud, ale nějakým způsobem je budu seskupovat, abych nezahlcoval emaily a mobilní zařízení uživatelů.

Nejdříve bylo nutné přidat další modely a tabulky do databáze. Vytvořil jsem model *Device*, pro uchovávání zařízení uživatelů, *Message* pro uchovávání zpráv a *SystemMessage* pro seskupování zpráv a sledování odeslaných notifikací.

Pokračoval jsem vytvořením jobu, který v nastavené intervaly kontroluje, zda existují nějaké nové zprávy. Pokud ano, seskupí je do jedné systémové zprávy, kterou odešle ve formě notifikace a emailu uživatelům. Stávající joby jsem také poupravil, aby neodesílaly emaily přímo, ale aby místo toho pouze vytvořily novou zprávu.

Samotné odesílání jsem vyřešil pomocí gemu **fcm**. Ten využívá Firebase Cloud Messaging od Google, což je multiplatformní řešení pro zasílání zpráv [20].

```
...
toks.in_groups_of(1000, false) do |tokens|
      fcm = FCM.new(FCM_KEY)
      options = {
        'data': {
            'title':     I18n.t('system_messages.mail_title'),
```

```
            'body':      I18n.t('system_messages.notification_body', count: self
                .num_of_messages),
            'msg_id':    self.id
          }
        }
        fcm.send(tokens, options)
end
...
```

Výpis 11: Ukázka kódu zaslání notifikace pomocí fcm gemu

Posledním stádiem bylo přidání podpory, do již vytvořeného API pro registraci mobilních zařízení uživatelů a pro možnost zapnutí nebo vypnutí notifikací.

4.8 Řešené problémy

V průběhu implementace funkcí a programování se vyskytlo mnoho problémů. Nakonec se mi podařilo všechny vyřešit za pomoci dokumentace, hledání na internetu, dotazováním se na fórech a konzultací s konzultantem z firmy. Zde uvádím ty, které mi zabraly dlouhou dobu k vyřešení.

4.8.1 Sidekiq namespace error

Velmi dlouho řešeným problémem, byla stále se objevující chyba v Sidekiqu v Sentry *NameError: uninitialized constant RateDownloaderJob*. Zajímavé na této chybě bylo, že jsem nikde neměl vytvořenou třídu RateDownloaderJob a ani žádný gem, související se Sidekiqem, tuto třídu neobsahoval. Navíc se chyba vyskytovala pouze v produkční verzi a nijak neovlivňovala funkčnost aplikace. Po dlouhém hledání a mnoha konzultacích s konzultantem práce jsem napsal na Github stránku k Sidekiqu o radu, kde mi však také nebyli schopni pomoci. Poté jsem zkusil přepsat Sidekiq workery do zpožděných jobů v railsech, ale chyba se objevovala stále. Nakonec jsme s vedoucím práce přišli na chybu.

Ukázalo se, že na produkčním serveru běží ještě jedna aplikace využívající Sidekiq. To je problém, protože pro použití více aplikací využívajících Sidekiq na jednom serveru je nutno použít jmenný prostor, aby se workery rozlišily a nemíchaly se do front z jiných aplikací. Díky tomu, že Sidekiq využívá pro spravování fronty Redis, bylo nutné použít **redis-namespace** gem, který zajistí unikátní jména jobů jednotlivých aplikací, běžících na stejném serveru. Po přidání a nastavení gemu byla tato chyba vyřešena.

4.8.2 IPv6 ping error

Při testování ping IPv6 adres se stále ukazovala chyba *can't modify frozen String (RuntimeError)*. Nejdříve jsem hledal chybu u sebe v kódu, ale po delším zkoumání se ukázalo, že se jedná o chybu v Ruby ve verzi 2.3.0. K vyřešení tohoto problému jsem musel aktualizovat Ruby verzi

projektu na 2.3.1. Tu stačí uvést v Gemfile pro daný projekt a spustit *bundle install*. Jelikož se jednalo o verzi, která pouze opravuje chyby, nebyla změněna žádná funkcionalita a problém byl odstraněn.

4.9 Znalosti získané při tvorbě tohoto projektu

Nakonec jsem splnil a implementoval vše, co se po mně v zadání chtělo. V této kapitole jsem se pokusil shrnout nejzajímavější problémy, které jsem na tomto projektu řešil. Samozřejmě jsem použil nepřeberné množství gemů a řešil daleko více problémů, ale popisovat všechny by bylo vzhledem k rozsahu práce kontraproduktivní.

Aplikaci nyní firma používá interně pro sledování svých vlastních projektů, stavů serverů a informování, když některá ze služeb vypadne. Jako další krok je v plánu rozšíření pro další společnosti, kterým by se podobná kontrolní aplikace také hodila.

Tento nejrozsáhlejší projekt mi poskytl i nejvíce znalostí. Naučil jsem se, co jsou Delayed job v RoR a jak s nimi pracovat, dozvěděl jsem se, jak vytvořit a psát API, a jak testovat aplikace. Zjistil jsem, jak řešit stránkování a vyhledávání v aplikaci. Naučil jsem se pracovat s Devise a mnoho dalšími používanými gemy, se kterými jsem se setkal poprvé. Za celou dobu praxe mi byla právě práce na tomto projektu největším přínosem.

5 CRM systém pro projektovou kancelář Kraje Vysočina

Mým druhým projektem, který jsem dostal na praxi zadaný, bylo vytvoření demo CRM systému, podle požadavků projektové kanceláře Kraje Vysočina. Tvorba této aplikace byla náročnější kvůli zpočátku nejasně stanoveným požadavkům ze strany kanceláře, a také kvůli poměrně krátkému času, určenému na dokončení demo verze.

5.1 Zadání práce

Zadáním bylo vytvořit demo CRM systém, který je určen pro řízení projektů (zakázek), obsahuje adresář kontaktů (zákazníci, dodavatelé a jejich kontaktní osoby), umožňuje vytvářet cenové nabídky ze vzorových kalkulací, úkoluje pracovníky, eviduje provedené práce a náklady (cesty) a dává přehled o průběhu práce na projektu. Systém obsahuje evidenci docházky včetně plánování dovolené. Dostupný je ze sítě internet bez nutnosti instalovat zvláštní programy (cloud řešení). Pro práci se systémem stačí webový prohlížeč a je použitelný také z mobilních zařízení (telefon, tablet, ...).

Úkolem bylo během šesti týdnů zprovoznit demo verzi systému, na základě které se kraj rozhodne, jaké firmě přidělí tuto zakázku. Díky tomu nebylo nutné implementovat kompletně všechny funkce, ale na některý místech stačilo nastínit, jak bude systém fungovat ve finální verzi.

5.2 Analýza zadání

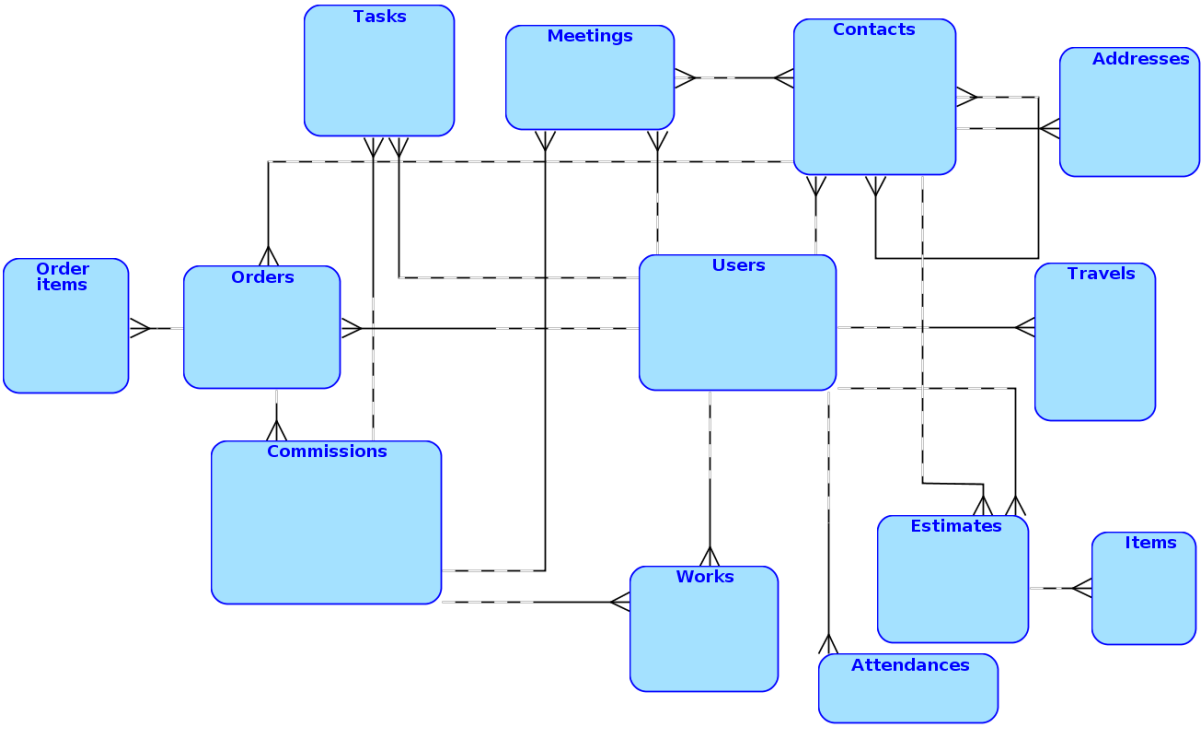
S konzultantem mé odborné praxe z firmy jsme si prošli požadavky projektové kanceláře a nastínili možná řešení. Následně jsem navrhl entity a vztahy mezi nimi. Kvůli úspoře času jsem pro vzhled mírně poupravil Bootstrap šablonu, kterou jsem měl již vytvořenou z RfCheck aplikace.

5.3 Modely a M:N vztah

Jelikož se jednalo o demo aplikaci, byly u všech modelů použity pouze základní validace testující přítomnost a délku zadávaných atributů. Zde jsem se také poprvé setkal s problémem implementace vztahu M:N mezi modely *Contact-Contact* a *Contact-Meeting*, neboť každý kontakt mohl mít více přidružených kontaktů a každý kontakt mohl mít více mítinků, stejně jako mítinku se mohlo účastnit více kontaktů.

V Rails aplikaci se tento vztah mezi modely tvoří pomocí *has_many :through* asociace. V databázi vytvoříme pomocnou tabulku a pomocí této asociace řekneme modelu, jaká tabulka je pomocná a udržuje nám informace o vztazích.

```
create_table "contact_contacts", id: false, force: :cascade do |t|
  t.integer "contact_id"
  t.integer "target_contact_id"
```



Obrázek 6: Konceptuální model CRM systému

```
    t.index ["contact_id", "target_contact_id"], name: "
        index_contact_contacts_on_contact_id_and_target_contact_id", using: :
        btree
    t.index ["contact_id"], name: "index_contact_contacts_on_contact_id", using
        : :btree
    t.index ["target_contact_id"], name: "
        index_contact_contacts_on_target_contact_id", using: :btree
  end
```

Výpis 12: Vytvoření pomocné tabulky contact_contacts pro vztah M:N

```
class ContactContact < ApplicationRecord
  belongs_to :contact
  belongs_to :target_contact, class_name: "Contact"
end
...
class Contact < ApplicationRecord
  ...
  has_many :contact_contacts, dependent: :destroy
  has_many :contacts, through: :contact_contacts, source: :target_contact
  ...
```

```
end
```

Výpis 13: Propojení pomocné tabulky s modelem Contact

Tímto docílíme vytvoření vztahu a umožní nám to používat AciveRecord metody jako např. *Contact.contacts* - vrátí kontakty, které patří pod daný Kontakt.

5.3.1 Internacionalizace - I18n

I18n gem, je určen pro poskytování multijazyčné podpory pro aplikaci. Je dodáván společně s RoR a snaží se poskytnou snadno použitelné nástroje pro správu jazyků [21]. Hlavní funkce I18n jsou:

- poskytování podpory pro angličtinu a další jazyky
- lokalizace dat a časových objektů
- usnadnění přizpůsobení pro ostatní jazyky
- vyhledávání textových překladů

Nejprve bylo nutné nastavit jazyk aplikace v *config/application.rb* na češtinu.

```
config.i18n.default_locale = :cs
```

Výpis 14: Nastavení lokalizace

Následně stačilo vytvořit soubory pro jednotlivé lokalizace a gem sám se již postaral o zobrazení správného jazyka. Soubory s překlady se obvykle nachází v *config/locales/* a mají koncovku .yml např. cs.yml, en.yml atd. Soubor může vypadat například následovně:

```
cs:
  error: "chyba"
```

Výpis 15: Ukázka yml souboru

V RoR aplikaci lze potom překlad volat pomocí metody *t*, která najde překlad v odpovídajícím yml souboru, podle aktuálního nastavení jazyka.

```
%h1= t('error') # prelozi se na <h1>chyba</h1>
```

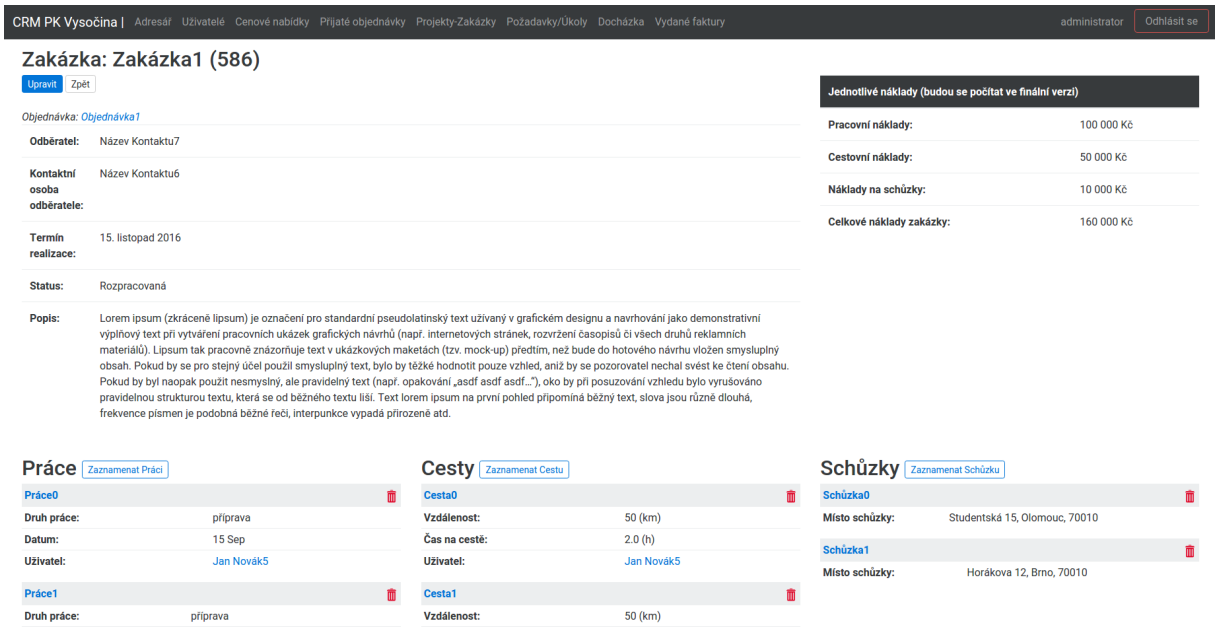
Výpis 16: Ukázka překladu ve views

5.4 Finální funkce CRM systému

Ve finální verzi se mi podařilo mnoho požadavků na CRM systém splnit. Bylo možno se přihlásit, zadávat kontakty, zakázky, položky k zakázkám, byl implementován víceúrovňový systém

pro tvorbu zakázky, aplikace si pamatovala přihlášení uživatelů a docházku a také zasílala informační emaily odpovědným osobám pro potvrzení.

Naopak nebyly implementovány některé funkce jako export do souboru, automatické počítání nákladů na zakázku a další funkce, které vyžadovaly upřesnění ze strany zadavatele a nebo by byly pro demo ukázku zbytečné, popř. příliš časově náročné.



Obrázek 7: Vzhled aplikace CRM Systému

Projevila se zde také vlastnost RoR frameworku, kdy je možné vyvinout a dodat funkční aplikaci za poměrně krátké časové období.

Na tomto projektu jsem si také uvědomil, jak je velmi důležitá přesná specifikace zadání a komunikace s klientem.

6 TopZájezdy

Poslední úkol, který jsem měl v rámci odborné praxe, byla výpomoc na projektu topzájezdy. Mělo se jednat o portál, který by umožnil jednoduché vyhledání a srovnání zájezdů různých cestovních kanceláří. Zjednodušeně to měl být portál podobný stránce Heureka, který by ale místo zboží z různých e-shopů srovnával zájezdy.

6.1 Zadání práce

Hned prvotním problémem u tohoto úkolu bylo, že každá cestovní kancelář má vlastní API a formát pro ukládání dat o zájezdech, popř. žádné API a data neposkytuje. Proto jsem nejdříve musel navrhnout obecný JSON formát, ve kterém bychom ukládali syntakticky analyzovaná data, které získáme od různých cestovních kanceláří, a které jsou pro nás relevantní. Následně jsem měl najít možnosti zpracování a importu velkého XML souboru od CeSYS.

6.2 CeSYS

Jedná se o systém pro prodej zájezdů na internetu. CeSYS shromažďuje informace o všech cestovních kancelářích, které ho podporují. Následně poskytuje XML soubor, kde jsou data o jednotlivých kancelářích a jejich aktuálních zájezdech. V dokumentaci k CeSYS systému jsem našel strukturu XML dokumentu a popis jednotlivých elementů a atributů. Na základě toho jsem navrhl obecný předpis pro JSON, který bychom použili jako výchozí, pro import dat.

```
"agency_id": "",
"tours": [
  {
    "code": "",
    "name":"",
    "descriptions" : [
      {
        "name": "",
        "description": "",
...
```

Výpis 17: Ukázka navrženého JSON formátu

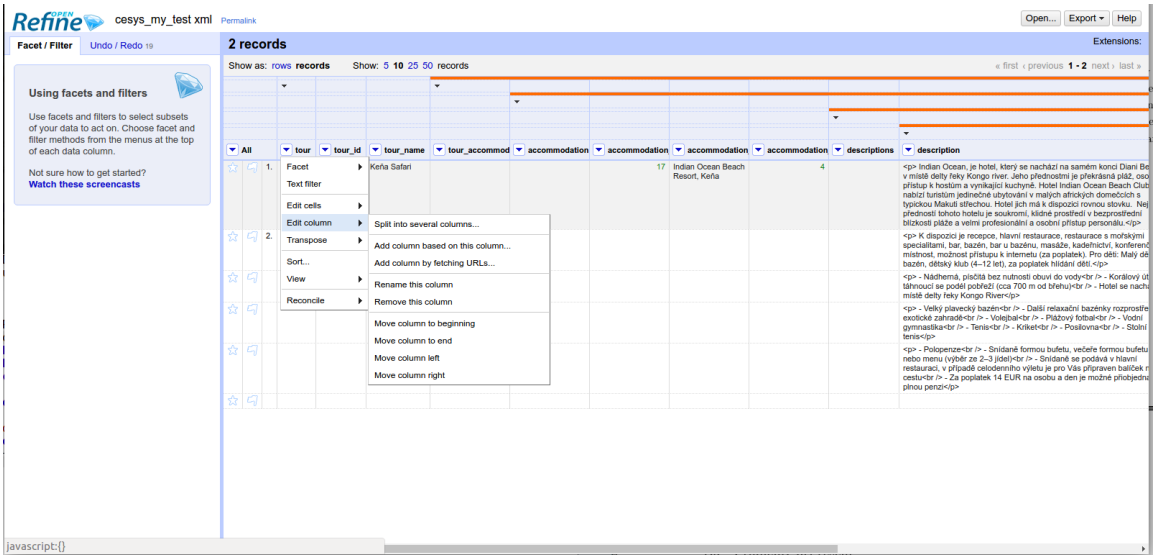
Musel jsem počítat také s tím, že XML soubor od CeSYSu má velikost minimálně 500 MB, takže se jedná o enormní množství dat, která se budou muset zpracovat [22].

6.3 Práce s OpenRefine

Mým druhým úkolem na tomto projektu bylo nastudovat dokumentaci k programu OpenRefine a zjistit, zda by se dal použít k automatizovanému zpracování a exportu dat z CeSYS systému do mnou navrženého JSON formátu.

OpenRefine je zdarma dostupný, výkonný open-source program pro práci s neuspořádanými nebo špatně uspořádanými daty, který je naprogramovaný v Javě. Dříve byl vyvíjen společností Google, která ho později uvolnila pod licencí open-source. Jeho další výhodou je, že dovede pracovat s velkými objemy dat, která dovede třídit a zpracovávat [23].

S OpenRefine se pracuje primárně pomocí GUI, které svým rozložením mírně připomíná tabulkové procesory jako Microsoft Excel nebo LibreOffice Calc a běží ve webovém prohlížeči. Na rozdíl od těchto programů však poskytuje daleko větší možnosti třídění a pokročilých funkcí pro práci s daty a je zaměřený na jinou oblast, než je kancelářská práce.



Obrázek 8: Ukázka GUI aplikace OpenRefine

Všechny úpravy a funkce prováděné na datech lze exportovat a následně se dají znovu aplikovat pro zpracování dalších dat. Naším cílem bylo nahrát CeSYS XML soubor do OpenRefine, upravit data do námi požadované podoby a vyexportovat je do mnou definovaného JSON formátu. To vše mělo probíhat zcela automatizovaně, právě díky možnosti exportu úprav. Tohoto se mi povedlo docílit pomocí gemu **refine-ruby**. Ten dovede volat funkce a komunikovat s programem OpenRefine přímo pomocí jazyka Ruby.

Nejdříve jsem v OpenRefine vytvořil nový projekt, nahrál do něj data a provedl na nich požadované úpravy. Historii úprav a aplikovaných funkcí jsem následně exportoval do souboru *operations.json*. Dále jsem vytvořil script v Ruby, kde jsem specifikoval soubor, který se má zpracovávat, soubor s operacemi, které se na nich mají provést a výstupní soubor, kam se mají zpracovaná data uložit.

```
...
prj = Refine.new({"file_name": 'cesys_example.xml' })
prj.apply_operations('operations.json')
p prj.export_rows('json')
extracted_data = prj.export_rows()
...
```

Výpis 18: Ukázka Ruby skriptu pro zpracování dat pomocí OpenRefine

Nakonec se mi povedlo pomocí skriptu automatizovaně upravovat CeSYS XML data do definovaného tvaru.

Problém však nastal při exportu do mnou navrženého formátu. Ukázalo se, že OpenRefine je v možnosti exportu zpracovaných dat poměrně omezený, především co se týká exportu do JSON formátu, protože neumí vnořovat data. Po řadě zkoušení a pročítání dokumentace jsem došel k závěru, že nelze exportovat data do mnou vytvořeného JSON formátu a museli bychom použít jiné formáty pro export např. do CSV, TSV apod. Z tohoto důvodu, jsme nakonec tuto možnost zpracování dat nezvolili.

6.4 Kiba a ETL

ETL se v počítačové vědě používá ke zpracovávání dat. Proces začíná extrahováním dat z heterogenní či homogenní datové struktury; následuje transformace dat, kdy jsou data vhodným způsobem upravena(transformována) pro uložení v požadovaném formátu či struktuře; posledním krokem je načtení dat, kdy jsou data načtena do finální databáze. Jelikož veškeré tři kroky mohou být časově náročné, provádějí se paralelně tzn. zatímco jsou data zpracovávána, extrahují se další data a zároveň se transformovaná data načítají do cílové databáze.

Kiba je jednoduchý ETL framework napsaný v Ruby a dostupný jako gem. Prostudováním dokumentace jsem zjistil, že stačí definovat zdrojový a cílový soubor a následně implementovat metodu *transform*, která určí, jak se data budou převádět. Pro přehlednost jsem každý jednotlivý proces ETL rozdělil do tří tříd CesysSource, CesysTransform a CesysDestination. Hlavní práci potom dělá třída CesysTransform, která převádí data do námi požadovaného formátu.

```
...
source CesysSource, input_file
transform CesysTransform
destination CesysDestination, output_file
...
```

Výpis 19: Ukázka souboru transform.etl

6.4.1 Nokogiri a zpracování XML

Nokogiri je gem pro zpracovávání dat z XML a práce s ním je velmi jednoducxhá [26].

```
...
xml = Nokogiri::XML.parse(File.open(input_file))
...
def process(xml)
    tours = []
    xml.css('tour_operator').each do |t|
      tour = {}
      tour[:agency_id] = t['id']
      tour[:tours] = import_accommodations(t)
      tours << tour
    end
    tours.to_json
  end
...
```

Výpis 20: Ukázka metody process z třídy CesysTransform

Metodu process volá Kiba automaticky. Zde se již soubor zpracovává pomocí Nokogiri. Kód je snadno čitelný, pro každou cestovní kancelář načtu jednotlivá ubytování a dále se vnořuji do XML dokumentu, dokud nenačtu všechna požadovaná data. Metoda *css* nalezne postupně všechny uzly v daném dokumentu. Výsledná data ukládám do hash struktury, v potřebném formátu, a kterou nakonec převedu do JSON pomocí *to_json* metody. Takto se mi povedlo převést data z CeSYS do požadovaného JSON tvaru.

Problémem se zde nakonec ukázalo Nokogiri, jelikož si v paměti vytváří celou stromovou strukturu zpracovávaného dokumentu a zabírá velmi mnoho místa v operační paměti. To by se následně projevilo například při zpracování dvou a více souborů o velké velikosti, kde by došla operační paměť na serveru a následně by došlo ke zpomalení zpracovávání dat.

Tabulka 3: Srovnání Nokogiri DOM a Nokogiri SAX parsování

Parser	**Počet položek**	**použitá RAM**	**Čas potřebný k zpracování (sekundy)**
Nokogiri DOM	500 000	1.0 GB	4.1
Nokogiri SAX	500 000	238 MB	12.9
Nokogiri DOM	1 000 000	2.0 GB	8.1
Nokogiri SAX	1 000 000	243 MB	25.5
Nokogiri DOM	2 000 000	3.2 GB	55.1
Nokogiri SAX	2 000 000	250 MB	16.6

Jako možné řešení jsem objevil parsování pomocí SAX, které Nokogiri také umí. Díky tomu by se enormně snížilo využití operační paměti, ovšem rychlost zpracování by byla znatelně pomalejší. Nakonec jsme se prozatím rozhodli ponechat stávající implementaci i za cenu využití více paměti RAM (viz tabulka 3).

6.5 Znalosti získané při tvorbě tohoto projektu

Při práci na tomto projektu jsem si vyzkoušel, jak obtížné je zpracování velkých objemů dat a jak zdlouhavé může být hledání správného řešení. Dozvěděl jsem se zde o možných technikách zpracovávání nejen v Ruby, vyzkoušel jsem si parsování XML a naučil jsem se pracovat s novými nástroji.

7 Závěr

Ze studia na vysoké škole jsem nejvíce využil znalosti, týkající se tvorby informačních a databázových systémů, skriptovacích jazyků a objektových principů programování.

Na praxi jsem se potom naučil práci se systémem správy verzí GIT nebo nové příkazy v příkazovém řádku Bash. Dále jsem přišel do styku s novými technologiemi a postupy tvorby softwaru a zdokonalil jsem se v hledání chyb a problémů. Zkusil jsem si, jaké je to pracovat ve firmě na reálných projektech a spolupracovat s ostatními členy týmu. Především jsem si však osvojil nové znalosti v programovacím jazyce Ruby a naučil se pokročilý vývoj webových aplikací v Ruby on Rails.

Během praxe jsem přispěl firmě svou prací na všech, mnou řešených projektech, nejvíce pak na RfCheck aplikaci, kterou nyní firma využívá pro sledování svých služeb a serverů.

Celkově bych tuto odbornou praxi hodnotil jako velmi přínosnou, ať již z pohledu získaných znalostí a zkušeností nebo ověření svých dovedností v praxi. Mnoho z nich každopádně ještě v budoucnu využiji a pomohou mi při uplatnění na trhu práce.

Tomáš Král

Literatura

[1] *Uživatelská příručka k definici malých a středních podniků*. Úřad pro publikace Evropské unie [online]. 2015, [cit. 2017-01-10]. Dostupné z: http://ec.europa.eu/DocsRoom/documents/15582/attachments/1/translations/cs/renditions/native

[2] *Firma Railsformers s.r.o.* Railsformers s.r.o. [online]. Ostrava, 2017 [cit. 2017-01-15]. Dostupné z: https://railsformers.com/reference

[3] *Oficiální stránka jazyka Ruby*. Ruby community. [online]. [cit. 2017-04-12]. Dostupné z: https://www.ruby-lang.org/

[4] *Oficiální stránka frameworku Ruby on Rails*. [online]. [cit. 2017-04-16]. Dostupné z: http://rubyonrails.org/

[5] *Webová stránka všech gemů Ruby on Rails*. RubyGems [online]. [cit. 2017-03-11]. Dostupné z: https://rubygems.org/

[6] ČADA, Ondřej. *Objektové programování: naučte se pravidla objektového myšlení*. 1. vyd. Praha: Grada Publishing, 2009. 200 s. ISBN: 978-80-247-2745-5.

[7] FOWLER, Martin. *Destilované UML*. Přeložil Martin PAVÍČEK. 1. vyd. Praha: Grada Publishing, 2009. 173 s. ISBN: 978-80-247-2062-3.

[8] *Iterativní vývoj*. Polytechnique Montreal [online]. [cit. 2017-04-17]. Dostupné z: http://www.upedu.org/references/bestprac/im_bp1.htm

[9] *Informace k HAML formátu*. The Haml Team [online]. [cit. 2017-03-11]. Dostupné z: http://haml.info/

[10] *Stránky SublimeText Editoru*. [online]. [cit. 2017-04-18]. Dostupné z: https://www.sublimetext.com/

[11] *Informace o IDE RubyMine*. JetBrains [online]. [cit. 2017-04-16]. Dostupné z: https://www.jetbrains.com/ruby/features/

[12] HARTL, Michael. *Ruby on Rails Tutorial: Learn Web Development with Rails*. 3. vyd. Addison-Wesley Professional, 2015. 744 s. ISBN 978-0134077703.

[13] *Dokumentace ke gemu Devise*. Github [online]. [cit. 2017-03-11]. Dostupné z: https://github.com/plataformatec/devise/wiki

[14] *Dokumentace k Sidekiq gemu*. Github [online]. [cit. 2017-03-11]. Dostupné z: https://github.com/mperham/sidekiq/wiki

[15] *Webová stránka projektu Sentry*. Functional Software, Inc. [online]. [cit. 2017-03-11]. Dostupné z: https://sentry.io/

[16] *Dokumentace ke gemu Grape*. Github [online]. [cit. 2017-03-11].
Dostupné z: https://github.com/ruby-grape/grape

[17] *Dokumentace ke gemu RSpec*. RSpec [online]. [cit. 2017-03-11].
Dostupné z: http://rspec.info/documentation/

[18] FEDUKE, Charles. *RSpec Test-Driven Development How-to*. 1. vyd. United Kingdom: Packt Publishing Ltd., 2013. 68 s. ISBN 978-1-78216-522-4.

[19] *Více informací o frameworku Sinatra*. [online]. [cit. 2017-04-18].
Dostupné z: http://www.sinatrarb.com/about.html

[20] *Informace o Google Firebase*. Google [online]. [cit. 2017-03-25].
Dostupné z: https://firebase.google.com/

[21] *Příručka k I18n gemu*. RailsGuides [online]. [cit. 2017-03-25].
Dostupné z: http://guides.rubyonrails.org/i18n.html

[22] *Oficiální stránky CeSYS projektu*. Darkmay s.r.o. [online]. [cit. 2017-03-25]. Dostupné z: https://www.cestovnisystem.cz/

[23] *Webové stránky projektu OpenRefine*. GithubPages [online]. [cit. 2017-03-25]. Dostupné z: http://openrefine.org/

[24] VERBORGH, R. - WILDE, M. - *Using OpenRefine*. 1. vyd. United Kingdom: Packt Publishing Ltd., 2013. 114 s. ISBN 978-1-78328-908-0.

[25] *Oficiální webové stránky frameworku Kiba*. Thibaut Barrere [online]. [cit. 2017-03-25]. Dostupné z: http://www.kiba-etl.org/

[26] *Dokumentace ke gemu Nokogiri*. Rubydoc [online]. [cit. 2017-03-25].
Dostupné z: http://www.rubydoc.info/github/sparklemotion/nokogiri